PROJECT : LARGE SCALE INFERENCE

Typo Correction : first and second order HMM

Florian Bertelli, Timothée Babinet, Ramine Hamidi

ABSTRACT

In the following document our aim is to realise an algorithm, based on HMM, to realise typo correction. In order to do this, we will use a first order HMM, and then a second order one.

Data description

- Data for this problem was generated as follows. The original document is the Unabomber's Manifesto, which was chosen not for political reasons but for its convenience of being available on-line and of about the right length. First, all numbers and punctuation were converted to white space and all letters converted to lower case. The remaining text is a sequence only over the lower case letters and the space character. Next, typos were artificially added to the data as follows: with 90 percent probability, the correct letter is transcribed, but with 10 percent probability, a randomly chosen neighbor (on an ordinary physical keyboard) of the letter is transcribed instead. Space characters are always transcribed correctly. In a harder variant of the problem, the rate of errors is increased to 20%.

1) First Order HMM

In this part, we've used a first order HMM, .ie each character can be corrected by looking at the character before and the current typed character. This model offers the following performances :

With 10% error rate :

- Char Tagging accuracy before : 89.82%
- Word Tagging accuracy before : 62.89%
- Char Tagging accuracy after : 93.21%
- Word Tagging accuracy after: 75.08%
- Error created by the model : 62.00
- Error corrected by the model: 41.61%

With 20% error rate :

- Char Tagging accuracy before : 80.59%
- Word Tagging accuracy before : 40.66%
- Char Tagging accuracy after : 86.87%
- Word Tagging accuracy after: 58.00%
- Error created by the model : 314
- Error corrected by the model: 42.02%

So, we can see that the order 1 model manages to correct 42% of the errors for the two different percentages, but creates more than 300 errors for the error rate of 20%, compared to only 62 for the error rate of 10%. This reveals that this model isn't efficient if we have many typos because it only takes into account the precedent char and so if you've already a mistake it will propagate easily.

2) Second Order HMM

In this second part, we've used a second order HMM, .ie each characters can be corrected by looking at the two characters typed before, and the current typed character. This model is based on the Viterbi algorithm and on the principle of HMM, but we've done some modification in order to adapt the order one algorithm.
Indeed, before the HMM of order 1 was represented by :

- init prob $\in \mathbb{R}^{|Y|}$: initial tag probabilities $p(y_0) = init\_prob[y_0]$
- transition prob $\in \mathbb{R}^{|Y|\times|Y|}$: tag transition probabilities $p(y_i|y_{i-1}) = transition\_prob[y_{i-1}, y_i]$
- observation prob $\in \mathbb{R}^{|Y|\times|X|}$: observation probabilities $p(x_i|y_i) = observation\_prob[y_i, x_i]$

To predict the character we want to find the argmax on y of:

$$p(y_1...y_n, x_1...x_n) = p(y_1)\prod_{i=2}^{n} p(y_i|y_{i-1})\prod_{i=1}^{n} p(x_i|y_i)$$

Now, we're working with the hypotheses of order 2 of HMM ie :

$$p(y_1 \dots y_n, x_1 \dots x_n) =$$

$$p(y_1)p(y_2|y1)\prod_{i=3}^{n} p(y_i|y_{i-1}, y_{i-2})\prod_{i=1}^{n} p(x_i|y_i)$$

As we can see now we have to compute $p(y_i|y_{i-1}, y_{i-2})$, this will be our new observation 3-dimensional matrix, .ie a tensor.

Thus, the HMM of order 2 will be represented by :

- init prob $\in \mathbb{R}^{|Y|}$: initial tag probabilities $p(y_0) = init\_prob[y_0]$
- transition prob $\in \mathbb{R}^{|Y|\times|Y|\times|Y|}$: tag transition probabilities $p(y_i|y_{i-1}, y_{i-2}) = transition\_prob[y_{i-2}, y_{i-1}, y_i]$

- observation prob $\in \mathbb{R}^{|Y|\times|X|}$: observation probabilities $p(x_i|y_i) = observation\_prob[y_i, x_i]$

Then as we want to get the argmax on y of :

$$p(y_1)p(y_2|y1)\prod_{i=3}^{n} p(y_i|y_{i-1}, y_{i-2})\prod_{i=1}^{n} p(x_i|y_i)$$

We will have to implement an adapted viterbi. Indeed, we've adapted viterbi in order to use the transition prob which is now in 3dimensions. Indeed, our aim is to maximise the equation mentioned above, by doing dynamic programming. So first we've initialized our backpointer and chart matrices with three dimensions ($26x26x26$) for (y, y_i, y_{i-1}) Then as before we fill the first column with the first observation and the initial probability. Then we do as viterbi for order 1 for the second column, because we don't have yet enough characters to do order 2. Then we've done it in a dynamic programming way. We find the *max* and *argmax* for each triplet (y, y_i, y_{i-1}), and fill the next column of the chart and the backpointer matrices. Finally, as for order one we've determined the best "path", and predict the most likely characters to put according to the two characters before, and the observed one.

Finally, we obtain the following performances :

With 10% error rate :

- Char Tagging accuracy before : 89.82%
- Word Tagging accuracy before : 62.89%
- Char Tagging accuracy after : 95.07%
- Word Tagging accuracy after: 81.48%
- Error created by the model : 62.00
- Error corrected by the model: 62.28%

With 20% error rate :

- Char Tagging accuracy before : 80.59%
- Word Tagging accuracy before : 40.66%
- Char Tagging accuracy after : 91.35%
- Word Tagging accuracy after: 71.93%
- Error created by the model : 313
- Error corrected by the model: 65%

We can see that the order 1 model manages to correct about 60 to 65% of the errors for the two different percentages, but creates more than 300 errors for the error rate of 20%, compared to only 80 for the error rate of 10%.
It is interesting to see that correcting just 2% characters more leads to an increase of the word tagging accuracy of more than 6%. This justify the importance of even small increases. This reveals to that our models is better to correct word with only one errors, than to correct word with two or more errors because of dependencies between characters into each word. If we compare the two different models ,ie order one and order 2 we have the following table :

	Error Corrected with 10 %	Error Corrected with 20 %	Char accuracy after with 10 %	Char accuracy after with 20 %
HMM Order 1	41.61 %	42.02 %	93.21 %	95.07 %
HMM Order 2	62,28 %	65 %	86.87 %	91.35 %

On this table we can see very well that the Order two HMM is more effective than the first order one. for 10% error rate the model 2 is better of 2%, but for the 20% error rate, the model 2 is better of more than 4%. This shows the robustness of the order 2 HMM, compared to the first order one.

3) Critics and evolution : noisy insertion of characters

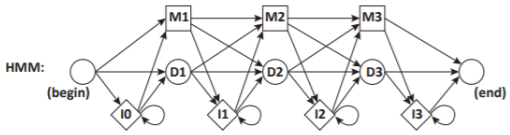
One problem of this problem is that it is only capable of taking into account the substitution characters, and didn't take into account the insertion of a new character into a word. For instance it won't be able to correct the word : "minee" into "mine" but it can correct "minw" into "mine". One way to be able to deal with this is to create a new data set, where you add a new possible tag which corresponds to an insertion of a char. In this approach you consider an insertion like another character. Your data could have this form for example :
[('s','s') , ('c', 'c') , ('c' , 'not') , ('r', 'r') , ('e', 'e') ,('a' ,'a') , ('m', 'm')]
If the real word is "scream" and the typed one is "sccream"

The aim of our HMM in this case will be to predict that the real character isn't 'c' but 'not' which corresponds to an unwanted addition of character.

4) Critics and evolution : deletion of characters

https://cme.h-its.org/exelixis/web/teaching/seminar2016/Example2.pdf

As described in this article, dealing with deletion is more complicated than dealing with addition, as deletion doesn't emit any symbol, you can't do as before, .ie adding a special tag corresponding to the deletion. We can deal with this problem by defining a new structure for the HMM:



With this structure, we can deal with insertion, and also deletion, because each state has now 2 new transitions : one for deletion and one for deleltion.

5) Unsupervised Learning

If we want to use unsupervised training, this means that we only have access to the typed text and not the real text. One way to deal with this problem is to use an external dictionary. Indeed, this is "kind" of what a human do to find the real word when there are typos. You check if the word is in "your dictionary", if not you first check the "closest" word, and if this isn't enough, you can also check the true meaning of the sentence.

We believe we can use do the same thing for an unsupervised model.
First, we check if the current word is in the dictionary, if yes we just consider it as true.
If it isn't in the dictionary, this means we've one typo or more. So, we now want to find the "true" word behind it, so we've to compute the similarity, or the distance between the word and each word of the dictionary. The most similar, or the nearest one will be the "true" word, and we will rewrite it.
We can wonder what distance will be the most effective, interesting, in this model. According to the literature, a good distance can be the

> Damerau{Levenshtein distance (named after Frederick J. Damerau and Vladimir I. Levenshtein[1][2][3]) is a string metric for measuring the edit distance between two sequences. Informally, the Damerau{Levenshtein distance between two words is the minimum number of operations (consisting of insertions, deletions or substitutions of a single character, or transposition of two adjacent characters) required to change one word into the other

If we use this distance, and a good dictionary we will be able to correct most words. However, this approach won't be able to understand the true meanings of the words, because this is a basic distance comparison, so we'll still have errors.

Conclusion

This project deals with HMM of order 1 and order 2, compares them, and then offers and suggests some ways to improve it to handle deletion or additions of characters. Furthermore, it discusses a way to implement an unsupervised algorithm, based on a dictionary, and the Damerau{Levenshtein distance.